

SDK C User Guide

V1.40

Table of Contents

Overview	3
Installation	4
Installing Visual C++ 2010 Express	4
Installing SDK C.....	4
CAN Interfaces	5
Other Installation	5
SDK Folder Layout	6
SDKC_ROOT/SDK Folder	6
Docs Folder	6
Platforms Folder.....	6
Portable Folder	6
Projects Folder	6
ProjectSamples Folder	6
ProjectTemplates Folder	6
Scripts Folder	6
Target Folder.....	6
Utilities Folder	6
Creating a Project with the Create Project Wizard	7
Starter Project Template Overview	8
File/Folder Layout	8
data Folder	8
source Folder	8
rootfs Folder.....	9
Screens	9
CAN Communications	9
UART Communications	10
Input/Output Support.....	10
Non-Volatile Storage.....	10
The PC Simulator	11
Compiling and Running.....	11
LCD	11
Save Screen	11
CAN	11

UART	12
The Target Hardware	13
Compiling	13
Creating a Settings Script	13
Creating a Splash Screen Bitmap	14
Target Update Packages	14
Building the Target Update Packages	14
Manufacturing Tool Full Update	15
USB Stick App Update	16
Utilities	17
Bitmap Conversion	17
CAN Log Converter	17
Multi Language Support	18
Prerequisites	18
Capabilities	18
Topology of the tool	18
Using the Spreadsheet	20
Generating source code	21
Output files	22
Language.inc	22
Font Customisation	22
Hardware Specification	24
Connector Pinout/Cable Harness	25
Cable 1	25
Cable 2	25

Overview

The development environment consists of two modes – one is when using the PC simulator to develop and test apps and the other is used to build the binary files that run on the target platform.

The fastest approach for developing apps is to use the PC Simulator. The PC Simulator allows the use of the one or two CAN Interfaces (CANdo or Kvaser supported) so the apps get access to “live” CAN bus. The PC Development Environment uses Microsoft Visual Studio 2010 as the IDE. The Express (Free) versions or the full versions can be used.

Once the app has been developed and tested on the PC, the Development Environment used for building the same app that will run on the target is the well known Eclipse IDE coupled with the Sourcery Codebench ARM compiler from Mentor Graphics.

Installation

To assist installation we have created a single .zip file and supporting batch files. Please follow the steps below.

Installing Visual C++ 2010 Express

If you already have an installed Visual C++ 2010 then you can skip this section.

1. Download Visual C++ 2010 Express from here <http://www.microsoft.com/visualstudio/en-us/products/2010-editions/visual-cpp-express>
2. Follow the instructions from Microsoft to install.

Installing SDK C

1. Download SDKC_Toolchain.zip file from here: http://www.cradl.co.uk/files/SDKC_Toolchain.zip.

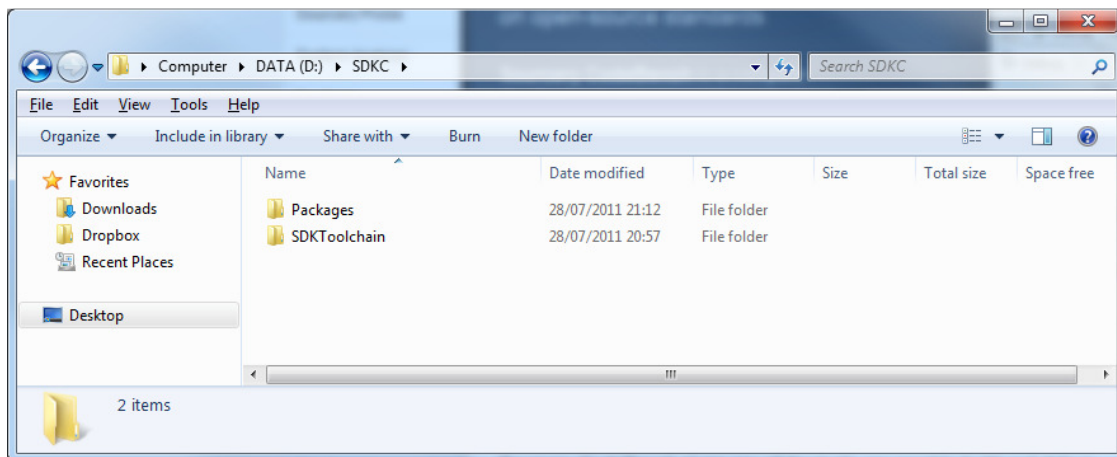
The file is around 200MB and should take < 10 minutes to download.

2. Create the root folder under which all the SDK files will be located. For example, this may be "D:\SDKC" as used in this installation guide. **Please note that in all further documentation, this SDK root folder will be referred to as SDKC_ROOT for simplicity.**

IMPORTANT NOTE

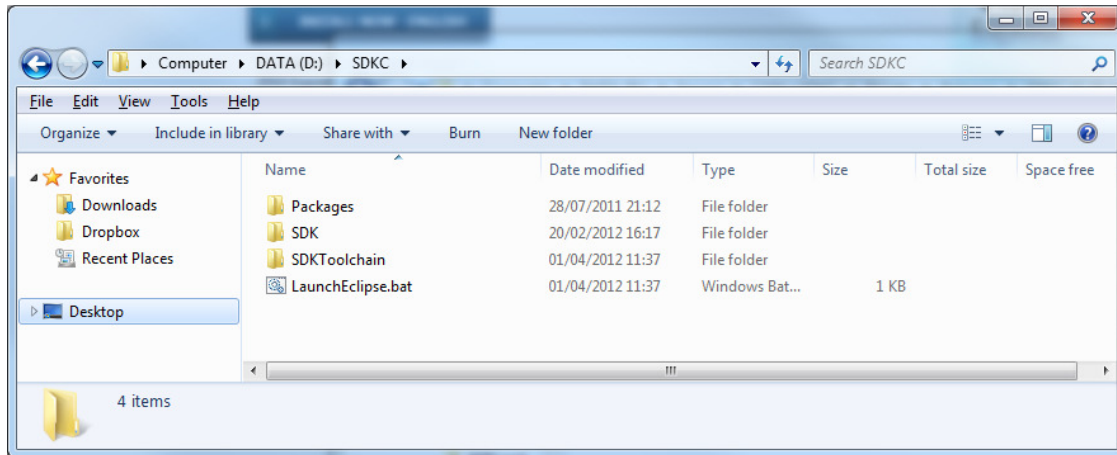
You MUST not have any spaces in any of the folder names above or below the SDKC_ROOT folder as this will cause the SDK to not function correctly!

3. Extract the SDKC_Toolchain.zip file into the SDKC_ROOT folder. You should then see two new sub-folders, SDKToolchain and Packages in your SDKC_ROOT folder as shown below.



4. Double click the SDKC_ROOT\Packages\SetupSDK.bat file. This will launch a new window and will setup the packages as required. Once it has finished press any key and the window will close.
5. Unzip the SDKC zip file into SDKC_ROOT\SDK.
6. If you purchased the optional J1939 Starter, unzip this into SDKC_ROOT\SDK\ProjectTemplates. You should then have a J1939Starter folder.

You should now have everything installed and your SDKC_ROOT folder (D:\SKDC in this example) should look like the figure below.



CAN Interfaces

The PC Simulator supports either the CANdo or Kvaser CAN interfaces.

For CANdo support, please download and install the drivers from <http://www.cananalyser.co.uk>

For Kvaser support, please download and install the drivers from <http://www.kvaser.com>

Other Installation

The BitmapTool.exe tool (located under SDKC_ROOT/SDK/Utilities) requires the Microsoft .NET Framework Version 2.0 Redistributable Package, this appears to be included by default in Windows Vista and Windows 7 but not with Windows XP. To allow the tool to run on XP there is a download available from Microsoft here <http://www.microsoft.com/download/en/details.aspx?id=19>. After installing this it is also necessary to do a check for updates which then installs an update.

If you wish to run the PC Simulator on another machine which has not had Visual C++ 2010 installed on it you will need to download the Microsoft Visual C++ 2010 Redistributable Package (x86) from here <http://www.microsoft.com/download/en/details.aspx?id=5555>.

SDK Folder Layout

The root level of the SDK folder tree (from here on referred to as SDKC_ROOT) contains the following:-

- Packages - contains the packages used to install the SDK
- SDK - The main SDK source folder
- SDKToolchain - Toolchain and other tools required to build SDK apps
- LaunchEclipse.bat - Special batch file that MUST be used to launch the Eclipse environment when building executable for the target hardware.

SDKC_ROOT/SDK Folder

The SDKC_ROOT/SDK folder contains the following:-

Docs Folder

This folder contains all of the documentation for the SDK (including this document).

Platforms Folder

This folder contains the libraries and header files that are platform dependent.

Portable Folder

This folder contains the libraries and header files that are not platform dependent.

Projects Folder

This folder is where your projects are stored. You should only be editing files under this folder! Please see the Starter Project section for details of what is in the project.

ProjectSamples Folder

This folder contains sample projects for your reference. Do NOT edit the sample projects in here, use the Create Project Wizard to copy them into your Projects folder where you can freely edit and customise them.

ProjectTemplates Folder

This folder contains starting templates for your projects. Do NOT edit the templates projects in here, use the Create Project Wizard to copy them into your Projects folder where you can freely edit and customise them.

Scripts Folder

This folder contains various scripts such as the Create Project and Build Update Wizards.

Target Folder

The folder contains target hardware specific items such as the Manufacturing Tool, target specific scripts and root file system libraries.

Utilities Folder

This folder contains various useful utilities such as the bitmap and log converters.

Creating a Project with the Create Project Wizard

Projects for you to work on are located under the SDKC_ROOT/SDK/Projects folder. Initially, this folder is empty, you will need to create a new project using the Create Project Wizard. All projects created by this wizard will be placed under the SDKC_ROOT/SDK/Projects folder.

IMPORTANT NOTE

You should ONLY edit projects contained under the SDKC_ROOT/SDK/Projects folder. DO NOT edit projects under the SDKC_ROOT/SDK/ProjectTemplates or SDKC_ROOT/SDK/ProjectSamples folders! Use the Create Project Wizard to create a copy of a template or sample under the SDKC_ROOT/SDK/Projects folder.

To create an SDK C project, run the Create Project Wizard (CreateProject.bat file located under the SDKC_ROOT/SDK/Scripts folder). This will ask you to select the type of project you wish to create a new project from. The options are:-

1. An existing project - This allows you to create a new project from one of your existing projects located under the SDKC_ROOT/SDK/Projects folder.
2. A template project - This allows you to create a new project from one of the template projects located under the SDKC_ROOT/SDK/ProjectTemplates folder.
3. A sample project - This allows you to create a new project from one of the sample projects located under the SDKC_ROOT/SDK/ProjectSamples folder.

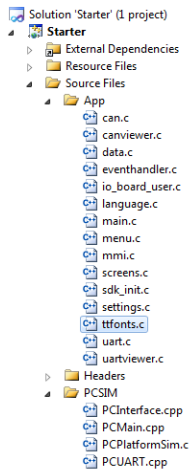
Once you have selected the source project type you will be asked to select the source project to copy from. You will then be asked to enter a name for the new project. After confirming your choices, your new project will be created under the SDKC_ROOT/SDK/Projects folder.

Starter Project Template Overview

This section describes the layout of the Starter template project. This is the project that we recommend you use to base your own projects on. Use the Create Project Wizard to create a copy of it into the SDKC_ROOT/SDK/Projects folder for you to work on.

File/Folder Layout

The top level project folder contains the Visual C++ 2010 (*.sln and *.vcxproj) and Eclipse (.project and .cproject) project/workspace files together with a few sub folders.



This screenshot from Visual C++ 2010 shows the C modules used in the project.

The project consists of the App source files (.c and .h files in the App and Headers section) and the PC Simulator files (in the PCSIM section). You should only modify the files in the App and Header sections.

The same source files (apart from the ones in the PCSIM section) are also included in the Eclipse project used when building the App for the Target.

data Folder

This folder contains the bitmaps and fonts that are used within the project. They are included in the project using the data.c/.h modules. You need to use the BitmapTool to convert bitmaps for use in the projects (see the Bitmap Conversion section for more information on the BitmapTool).

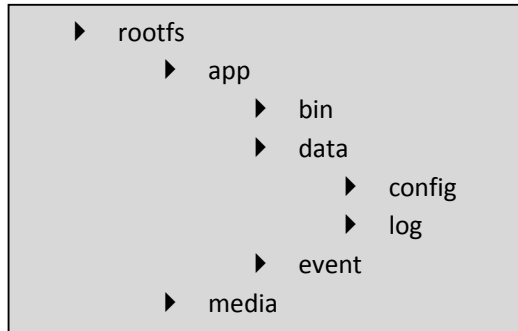
source Folder

This folder contains the main source code for the project.

- can.c/.h – support for the CAN Ports.
- canviewer.c/h - for viewing and recording raw CAN frames. Accessed from the Diagnostics menu.
- data.c/.h – contains the constant data used for the font and bitmap data (also data.h), the data files are in the data folder/sub-folders.
- eventhandler.c/.h - support for handling events from OS scripts.
- io_board_user.c/.h – support code for the integrated I/O sub-system.
- language.c/.h/.inc - contains the string data used for the text data. Please see the Multi Language Support section for more information on language support.
- main.c/.h – contains the main loop and also timer call back routines.
- menu.c/.h – example menu screens accessed by pressing and holding key 5.
- mmi.c/.h – functions to control the screen and keys – not usually modified.
- screens.c/.h – the user screens are defined here.
- sdk_init.c/.h – contains the start-up and system initialise code.
- settings.c/.h – has the settings structure that is saved to the EEPROM.

- ttfonts.c/.h - support for handling True Type Fonts.
- uart.c/.h – support for the UART Port.
- uartviewer.c/.h - for viewing raw UART data. Accessed from the Diagnostics menu.

rootfs Folder



The rootfs folder is an emulation of the part of the Linux root file system on the Target that is accessible by the App.

It is created by either running the PC Simulator or building the Target executable with the Eclipse IDE.

Its structure is shown in the figure opposite.

Screens

The screen handling and drawing functions are located in screens.c/.h.

Key functions are defined in ScreenEnter(), usually default key settings are loaded first followed by the particular requirement for the keys. The keys can have text or a bitmap image on them.

The screen background image/text is drawn in ScreenCreate(), this is the fixed part of the screen contents and is drawn in layer 1. This function is only called when the screen is initially displayed or when a complete screen redraw is required (by setting the FLAG_UpdateScreen variable to SET).

The foreground (changing part of the screen) image/text is drawn in ScreenUpdate() and is drawn every loop (about 40 frames per second). This needs to be erased before redrawing and should be drawn in layer 2.

To create bitmaps you need to do the following:-

- Draw the bitmap in a suitable graphics package – save as 24-bit-per-pixel image BMP.
- Use the BitmapTool.exe (located in SDKC_ROOT/SDK/Utilities) to convert the bitmap to source code.
- Copy the .c and .h files to the data Folder in your project.
- Edit data.c and data.h to add an entry in each for the new files added to the data folder.
- In the code, use the outimage() function to position and draw the bitmap on the screen.

To draw text to the screen:-

- Use outtextxy() or outnumxy() – both draw in the selected colour.

To draw graphics primitives to the screen:-

- Use any of the standard graphic primitives contained in SysLib.

CAN Communications

CAN port communications are handled in can.c/.h.

- Set up the CAN ports in the CANInit() function.
- To receive CAN frames, add support code in CANDecodeRxStd() and CANDecodeRxExt().
- To transmit CAN frames, use the CANSend() function. Use the CANIDEXT_CREATE macro in can.h to create an extended (29-bit) CAN Id.

UART Communications

UART communications are handled in uart.c/.h.

- Set-up port in the UARTInit() function.
- Modify the low level receiver if required. This is used to detect the start and end of a message.
- To transmit use UARTSend() function

Input/Output Support

I/O support is handled in io_board_user.c/.h.

- Configure inputs using bIO_User_Init()
- vSampleIOBoardInputSignals() is called every loop and polls the inputs and here is where the calibration is placed
- To output a signal use bSetIODigOutput()

Non-Volatile Storage

Data can be stored in non-volatile storage in the Target unit, this is achieved with an EEPROM emulated by a file stored in the NAND Flash. The EEPROM system is controlled by two SysLib functions, Restore_eeprom() and Store_eeprom(). In the Starter project the gpSettings structure is saved in EEPROM (see settings.c/.h).

The Restore_eeprom() function will load the data and check the CRC, if the CRC fails it will call load_eeprom_defaults() in your app code to set up the structure with default values.

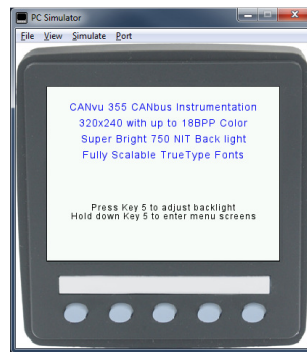
The Store_eeprom() function, which should be called every main loop from CANvuMainLoop(), will only try and save the data if the gucStoreEepromRequired global flag variable is set to SET or a non-zero value. If this variable is not set it will do nothing. If this variable is set, it will calculate a CRC checksum of the data and if the CRC does not match, the data will be saved. It will always clear the gucStoreEepromRequired to zero. In the Starter project, the vOneMinute() timer function sets this flag so that any changes to settings data are saved at a minimum period of 1 minute.

The PC Simulator

Compiling and Running

Once you have created a project with the Create Project Wizard, load Visual C++ 2010 and follow these steps:-

1. Open the project by selecting File->Open->Project/Solution from the menu.
2. Browse to the SDKC_ROOT/SDK/Projects/YourProjectName folder and double click the YourProjectName.vcxproj file to open it.
3. Build the project by selecting Debug->Build Solution from the menu.
4. Run the project in the debugger by selecting Debug->Start Debugging from the menu. The PC Simulator will run and you will see something like this:-



LCD

The Simulator provides a one to one mapping of the hardware LCD pixels to PC screen pixels. This means that the simulator provides an accurate representation of the LCD with no spurious artefacts or distortions. For this reason, the zoom factors are limited to whole numbers, 1:1, 1:2 and 1:4. At the 1:1 setting, the pixels are one to one, so the LCD window occupies 320 pixels in the X direction, by 240 pixels in the Y direction, matching the hardware LCD exactly. At the 1:2 setting, each LCD pixel is expanded to 2 pixels in the X direction & 2 pixels in the Y direction, hence each pixel occupies 4 pixels on the PC screen. At the 1:4 setting, each LCD pixel is expanded to 4 pixels in the X direction & 4 pixels in the Y direction, hence each pixel occupies 16 pixels on the PC screen.

Save Screen

The Simulator allows the screen to be frozen & saved to a bitmap file. This is useful during graphics development, as it allows screen bitmaps to be simultaneously compared in a bitmap editor. The screen bitmaps can also be useful for documentation purposes.

To save the current screen, select 'File->Save Screen' and specify a location and filename in the dialog box that appears.

The bitmap is a one to one representation of the simulator LCD screen, so if the 1:2 zoom factor is selected, then the bitmap will be twice the width & twice the height of the hardware LCD.

CAN

The CAN ports are simulated with one of the following options:-

- CANdo USB CAN Interface
- Kvaser CAN Interface
- A CAN log file

To select the driver, use the `can_select_driver()` function (as seen in the Starter projects' `can.c` `CANInit()` function). All the other CAN functions will now be directed through this driver.

UART

The UART serial port is also simulated by using a COM port on the PC. By default this feature is turned off, but may be enabled by selecting 'Port->UART' from the simulator menu & selecting the required COM port and clicking 'Open Port'. Please note that the baud rate and other parameters of the COM port are set from the parameters used in the source code.

The Target Hardware

Compiling

To compile your app for the Target, you need to use the Eclipse IDE. To run the Eclipse IDE, double click on the LaunchEclipse.bat file located under the SDKC_ROOT folder.

IMPORTANT NOTE

If this is the first time you have run the Eclipse IDE, there are a few settings that need to be made before we build a project. From the Window->Preferences menu, under the General->Workspace section make sure that the 'Build automatically' option is not checked and the 'Refresh automatically' and 'Save automatically before build' options are both checked.

To build your project please follow these steps:-

1. Select File->Import from the menu.
2. Under General, select Existing Projects into Workspace and click the Next button.
3. Click the Select root directory Browse... button, navigate to SDKC_ROOT/SDK/Projects folder and click the OK button.
4. Your project(s) should be shown in the Projects list with a tick against them. Click the Finish button. Your project(s) should appear in the Project Explorer pane.
5. In the Project Explorer pane, right click on the project you wish to build. From the context menu, select Build Configurations->Set Active->Release to select release build mode.
6. Right click on the project in the Project Explorer again and select Build Project.

Your Release executable should now be built and copied to the rootfs/app/bin project sub-folder.

Please note that Debug builds produce an executable with _Debug appended to the name. For example, if your project was called Test, the Debug build would produce an executable file called Test_Debug whereas the Release build would produce an executable with no extension called Test. All executable are copied to the rootfs/app/bin project sub-folder.

Creating a Settings Script

Each Target unit has the capability of auto-launching one of up to five apps stored on the internal NAND flash. Each app must have a unique file name and each app will have its own independent configuration file that is used to store non-volatile settings (simulates an EEPROM to store configuration and settings data).

The selection of which app to run is made by holding down one of the 5 keys during the boot-up sequence (while the splash screen is shown before an app starts). Once an app has been selected, this app then auto-launches during subsequent boot-ups unless another key is held to select an alternative app.

This feature allows one standard display to be configured to have alternative features depending on which app is currently active. This may also be used to access built in test functionality or product information in place of a user manual.

To control which apps are launched you need to create a settings.sh (case sensitive) file in the rootfs/app/bin folder (located under your project folder) which needs to look something like the following:-

```
#!/bin/sh
# The application launched when key 1 is held
APP1="YourApp"
# The application launched when key 2 is held
APP2="AnotherApp"
# The application launched when key 3 is held
APP3="YetAnotherApp"
# The application launched when key 4 is held
APP4=""
# The application launched when key 5 is held
APP5=""
```

Note that the only app that has to be present is APP1, the others are all optional. There is an example settings.sh file in the SDKC_ROOT/SDK/Target/examples/settings folder for you to copy and modify.

IMPORTANT NOTE

The settings.sh file MUST be a Linux compatible file with Linux line endings. DOS files have a carriage return + line feed (CR+LF, 0x0D+0x0A) at the end of each line. Linux files just have a line feed (LF, 0x0A) at the end of each line.

Creating a Splash Screen Bitmap

You also need to create a splash screen bitmap which will be displayed while the target unit is booting. The splash screen must be called splashscreen.bmp (case sensitive) and must be placed in the rootfs/app/bin folder (located under your project folder). There is an example splashscreen.bmp file in the SDKC_ROOT/Target/examples/splashscreen folder for you to copy and modify.

Target Update Packages

There are two methods of updating the target hardware units, these are as follows:-

- **Manufacturing Tool** - This method completely updates everything in the Target including the boot loader, OS, root file system and your app. *This method is recommend during production as this guarantees the correct version of the boot loader, OS and root file system are programmed into the NAND flash along with your app.*
- **USB Stick** - This method ONLY updates your app, the splash screen and any associated files, it DOES NOT update the boot loader, OS and root file system. *This method is recommended to update target units during development and when they are in the field.*

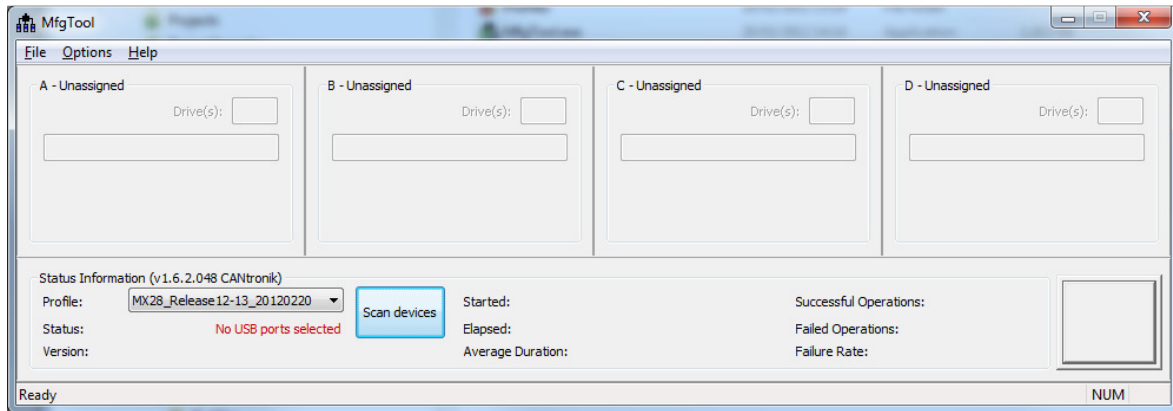
Building the Target Update Packages

To update the target unit, you need to build the Manufacturing Tool update packages. This is done by running the Build Update Wizard (called BuildUpdate.bat) under the SDKC_ROOT/SDK/Scripts folder. The Wizard will ask you which project you wish to create the update for before creating both

a new Manufacturing Tool Profile and a USB Stick Update. Once the updates are built, it will also tell you how much space you have used out of the total amount of space available for apps.

Manufacturing Tool Full Update

The Manufacturing Tool is located under the SDKC_ROOT/SDK/Target/mfgtool folder, it is called MfgTool.exe. When you run it you will see the following screen:-

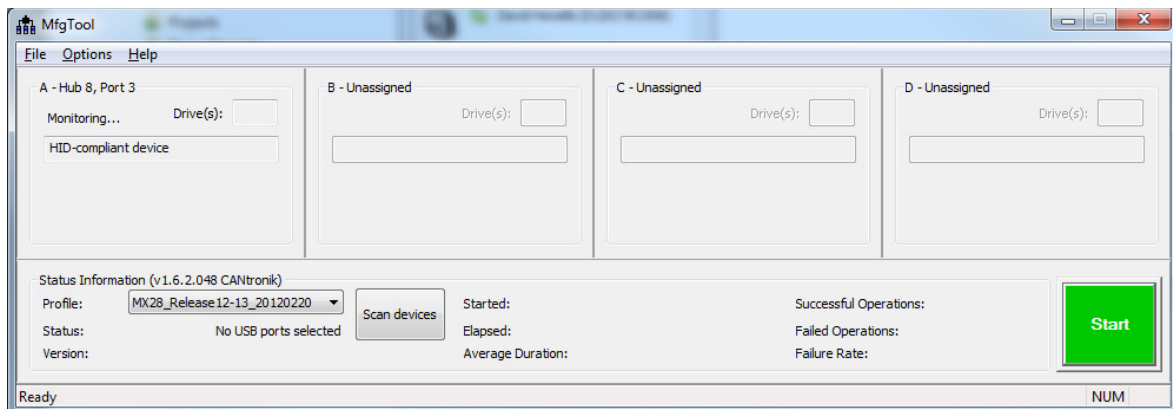


On the back of the Target unit there is a small flap, under which is a USB Type A Connector and a small button beside it. This will put the unit into USB boot mode, do the following:-

- With the unit powered off, hold down the small button
- Turn on the power to the unit while keeping the button held down
- Keep the button held down for at least 2 seconds before releasing it.

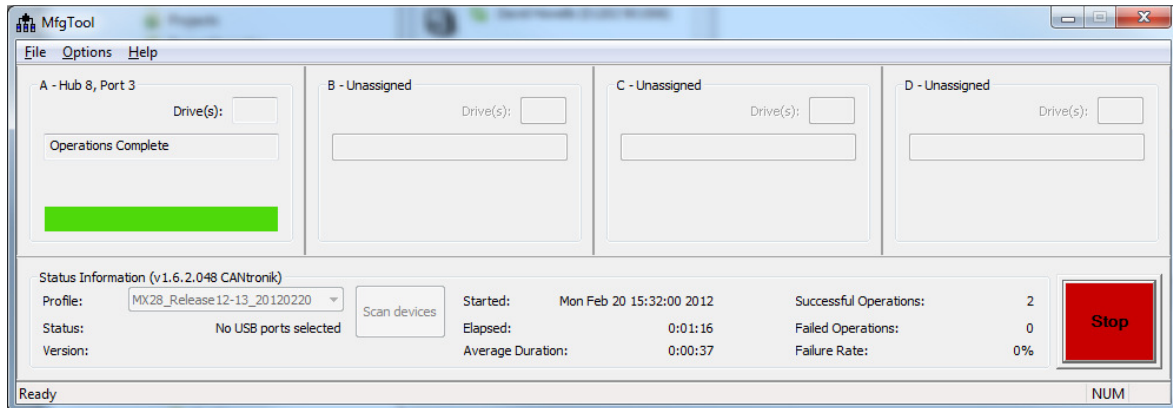
Now plug a USB Male Type A to USB Male Type A lead into the unit and also to the PC. Windows will install a standard HID driver for the unit, there are no special drivers needed.

Once Windows has finished installing the HID device driver, click the "Scan Devices" button. You should see the "A" slot change to "Monitoring" with a HID-compliant device and a green "Start" button will appear as shown on the following screen:-



From the "Profile" drop down, select the "MX28_Release13-13" profile (this is the most recent firmware release with a special test application in it) or the profile with the same name as your project and click the start button. The green "Start" button will change to a red "Stop" button and

the update procedure will begin. After approximately 40 seconds, the message "Operations Complete" should appear as follows:-



The update procedure is now complete and you may close the MfgTool and un-plug the unit from the USB port and turn of the power.

If you used the "MX28_Release13-13" profile, there is only a test app installed on the device. You may now put your app on the device by using the USB Stick Update procedure.

If you used the profile with the same name as your project then your app will already be installed and you do not need to use the USB Stick Update procedure.

USB Stick App Update

You will need a standard USB memory stick formatted with the FAT32 file system perform this type of update. Insert the memory stick into your PC and copy the contents of the usbupdate folder (located under your project folder) to the previously blank memory stick. **Please note that you should only copy the files from the usbupdate folder to the root of the USB Stick – not the usbupdate folder itself!**

Power up the Target hardware and wait for the installed app to run then plug the USB stick into the rear USB connector. The USB Stick will be automatically detected the update will begin. Once the update has finished, the backlight will flash on and off to indicate the process is complete. You should then remove the USB Stick and the unit will reboot. The unit will now be updated with your latest apps and by default will run APP1. Use the key hold mechanism while booting to choose an alternative app if required.

Utilities

Bitmap Conversion

The Bitmap Conversion tool (called BitmapTool.exe and located under SDKC_ROOT/SDK/Utilities) is used to convert bitmaps for use within projects. All apps require .c and .h files to allow bitmaps to be displayed. The user has to edit data.c/disp_ext.c and data.h/images.h and add the new .c and .h files for them to be used in a project. Please ask for information on how to add and use alternative true type fonts.

CAN Log Converter

The CAN Log Converter (called ctlogcvr.exe and located under SDKC_ROOT/SDK/Utilities) is used to convert different CAN logs to different format. Currently, CANtronik Binary (CTB), PCAN and CANalyzer formats are supported. Please run the program from the Command Prompt. The command line parameters are as follows:-

```
CTLOGCVT v2.01, Copyright (c) 2011 CANtronik Ltd

usage: ctlogcvr input_type input_filename.ext output_type output_filename.ext

input_type and output_type can be one of the following:-
    cantronik - CANtronik binary format (.ctb)
    canalyzer - Vector CANalyzer format (.asc)
    pcan      - Peak PCAN Explorer format (.trc)
```

Multi Language Support

The language spreadsheet is to be found in the SDKC_ROOT/SDK/Utilities folder, it is called Language_Table.xls. It is also used in all the starter template projects and is located under their source folders.

The spreadsheet is a development tool – it can process app text phrases and words entered in cells in the spreadsheet and generate the ‘language’ source files for the project.

Benefits of language source files generated from the spreadsheet tool are:-

- Provides a convenient method of editing app text in a ‘syntax free’ environment.
- No ‘C’ programming knowledge needed so is ideal for use by linguists/translators.
- Additional language provision in app just requires new columns in the table and translations to enter in the table.
- When your app makes use of the FreeType library capabilities of the SDK the table’s code generation handles appropriate encoding of special characters in the text data.

Note: In producing a code generation tool like the CANvuSDKC_Language_Table.xls it is not possible to take into account every possibility of the users app programming style or structure of existing apps written using a source code SDK. So while the code generated by the tool should be a ‘drop in and compile’ solution for typical projects, introduction of the auto-generated language source-code files to some existing apps may present compilation or link issues that will need to be manually resolved – this is beyond the scope of this guide.

Prerequisites

To use the Language Table spreadsheet you will need to have Microsoft Office Excel 2003 or later.

Capabilities

String data (words/phrases) in any language which can be written using characters lying within the Basic Multilingual Plane (BMP) of the Unicode standard can be entered and will be encoded by the tool.

The tool can handle string data for up to 23 languages – one per column starting in column C. Note that the language order in the sheet will (normally) define the language order presented in the language selection menu in your app.

The language list will be terminated by a column having no language name entry in its Row19 cell.

To make efficient use of memory in the target system, the C source code generated by the tool assigns each string’s data as an individual string constant. The ‘phrase’ by ‘language’ array(s) generated are then 2D array(s) of pointers to the discrete phrase/word data string constants.

The tool will process and encode up to 30 arrays (groupings) of string data.

Topology of the tool

The figure below shows the main elements of the language table spreadsheet:-

Language Table

Parameters :	
Name of the .h/.c/.ino files	language
Comment	Language definitions for SDK Starter project
Name for number of language	NB_LANGUAGE
Header files to #include :	
	one2.h
	display.h
	duklib.h

[Click here to generate C source code...](#)

Using UTF8 Strings

Source code definitions	English	Spanish	French	Chinese (Simplified)
#ucSystemWords[wordx_TEXT_MAX]				
#NONE				
#LANG	ENGLISH	ESPAÑOL	FRANCAIS	中文
#LANGUAGE	LANGUAGE	IDIOMA	LANGUE	语言
#CONFIG_MENU	CONFIG MENU	MENÚ DE CONFIG.	MENU CONFIG.	配置菜单
#SETTING_MENU	SETTING MENU	MENÚ DE AJUSTES	MENU REGLAGE	设置菜单
#SETTING	SETTINGS	AJUSTES	REGLAGES	设置
#BLEEP	BLEEP	BIP	BIP	蜂鸣
#RESETALL	RESTORE DEFAULTS	RESTORE DEFAULTS	REGLAGES USINE	恢复默认
#PROG_TRANSFER	PROG. TX	PROG. TX	PROG. TX	发送程序
#COM_VIEWER	COM VIEWER	INFO COM	INDICATEUR COM	通信器
#ABOUT	ABOUT	ABOUT	ABOUT	关于
#Sending_Prog	Sending Program	Enviar Programa	Envoi du Programme	发送程序
#Receiving_Prog	Receiving	recepción	Réception	接收程序
#_END				
};				

The main elements are:-

Red Zone:-

Phrase storage array names and labels that are to be used within the app to access the phrases and words.

White Zone:-

Phrases and words entered here, to the right of the red zone, are the strings that will be available for use within your app. Each column represents a language.

String format selection cell (light green):-

This cell is for selection of the format of string data storage when source code is generated.

'Click here..' button (green):-

Clicking on this button initiates the processing of the data and generation of the source code files that can be incorporated in your project.

Parameter zone (light blue):-

This area contains parameters that affect aspects of source code generation. You shouldn't normally need to make any changes to these.

Preprocessor zone (light orange) :-

This area, not shown in **Error! Reference source not found.**, may be used to provision for conditionally compiled code in your app. (e.g. for excluding or including phrases in the build of your project according to product build variant). Normally this area would be left blank.

Using the Spreadsheet

Your product requirements document or your project plan will probably set out the language requirements for your app.

Usage is probably best described by example...

We will consider here that you decide to start with a blank table – i.e. no phrases/words entered. (This would be like having selected cells C20..K119 and cleared the cell contents, and also clearing row C19..K19 of language names)

This example usage assumes English as the default language. We'll have the following words available for the app:-

Gallon, GallonUS, Hood, Trunk, Beep, Config Menu, On, Off, Enable, Disable, Language, Select, Reset All.

We'll build the table to provide support for English, English US, Russian, Chinese (Simplified)

1 – Add, along row 19, starting at C19, the languages that you are going to support in your product. In the accompanying example we will enter English(UK) in C19, English(US) in D19, Russian in E19, Chinese(Simplified) in E20.

2 - Add, the phrases/words of your default language, one per cell in its language column. Leave two blank cells above the first of your phrases/words.

3 - For each other language, enter translations of each phrase/word. (For this example the writer used <http://translate.google.co.uk/> to obtain translations of the example words/phrases).

4 – When your language table is populated you need to put some entries in the Red Zone – these will help to map your phrases/words to labels that you can reference them within your app code.

- i. Firstly we need to name the array that the strings will be accessed through in the app code.
In this example this is **cucSystemWords**. This must be followed in the cell below by a { symbol.
- ii. Each row containing a phrase/word must have a label in the Red Zone – these label words are the labels your app code will use to access the string data – for example wLANG, wLANGUAGE, wCONFIGMNU, wBEEP etc.
- iii. An extra label after the last phrase/word in the group is added, e.g w_SIZE. This serves to dimension the storage array in the generated language source code for your app.

- iv. The Red Zone cell below the last phrase/word entry of the current group (e.g. cucSystem Words in this example) must contain };

5 – If your app requires it, there can be additional groups of words, each group bounded and labelled as described in 4i to 4iv above.

The figure below shows one language and associated RedZone content.

Source code definitions	English(UK)
cucSystemWords	
{	
wLANG	English(UK)
wLANGUAGE	Language
wGALL	Gallon
wGALLUS	US Gallon
wCOLOUR	Colour
wHOOD	Bonnet
wTRUNK	Boot
wBEEP	Beep
wCONFIGMNU	Config Menu
wON	On
wOFF	Off
wENABLE	Enable
wDISABLE	Disable
wLANGUGAGE	Language
wSELECT	Select
wRESETALL	Reset All
w_SIZE	
};	

Generating source code

When you have completed your phrase/word entry in the sheet and have satisfied the requirements of the Red Zone described above, then you are ready to generate the source code to add to your app.

For normal usage, ensure that you have selected 'Using UTF8 Strings' in the light green cell below the 'Click here....' button.

Click on the 'Click here to generate C source code...' button.

The sheet will be processed and the output files will be generated.

The files will be in the same folder as the spreadsheet file was opened from. They will need to be moved to the 'source' subfolder of your project, if you didn't run the spreadsheet in that location.

Processed string data is encoded as UTF8 byte data in the generated C source code when using the normal 'Using UTF8 Strings' mode.

If the 'Using ASCII Strings' mode is selected the processed string data will be encoded as byte data in the generated C source code. In this mode any characters of codepoint greater than 127 will be

encoded as a '?' (question mark) character. You will get a warning dialog if this occurs and the first cell found to contain an out of range character will be set active.

Output files

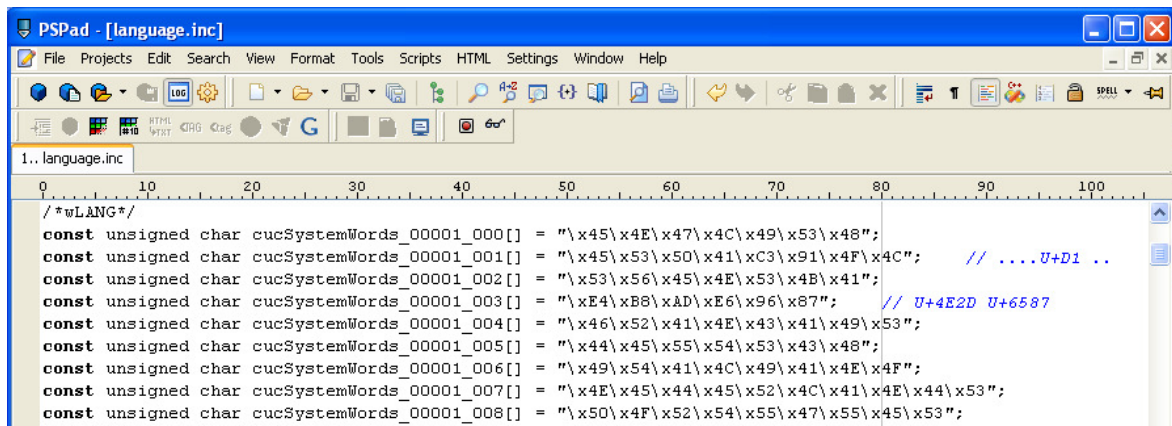
The tool will normally generate the following files:-

- Language.h
- Language.c
- Language.inc (#included by language.c)

Language.inc

Language.inc contains the encoded string constants data.

A feature of this file is the comments following string consts that contain characters of a codepoint above 127. These comments, when present, show the codepoint in U+nnnn form for these characters. Characters up to codepoint 127 in that string will be represented by a dot. The figure below illustrates this.



```
1.. language.inc
/*wLANG*/
const unsigned char cucSystemWords_00001_000[] = "\x45\x4E\x47\x4C\x49\x53\x48";
const unsigned char cucSystemWords_00001_001[] = "\x45\x53\x50\x41\xC3\x91\x4F\x4C"; // ....U+D1 ..
const unsigned char cucSystemWords_00001_002[] = "\x53\x56\x45\x4E\x53\x4B\x41";
const unsigned char cucSystemWords_00001_003[] = "\xE4\xB8\xAD\xE6\x96\x87"; // U+4E2D U+6587
const unsigned char cucSystemWords_00001_004[] = "\x46\x52\x41\x4E\x43\x41\x49\x53";
const unsigned char cucSystemWords_00001_005[] = "\x44\x45\x55\x54\x53\x43\x48";
const unsigned char cucSystemWords_00001_006[] = "\x49\x54\x41\x4C\x49\x41\x4E\x4F";
const unsigned char cucSystemWords_00001_007[] = "\x4E\x45\x44\x45\x52\x4C\x41\x4E\x44\x53";
const unsigned char cucSystemWords_00001_008[] = "\x50\x4F\x52\x54\x55\x47\x55\x45\x53";
```

The role of these comments is to help you to check that you have available in your font any 'special' characters used in the phrase/word data of your app.

Font Customisation

While FreeType font usage is covered in the SDK C Scalable Fonts Guide, it is probably appropriate to outline a method that you may use to get all characters that occur in your phrase/words language data into your font data. This method requires access to a font creation/ font editing tool. The SDK does not have such a tool. There are commercial and freeware font tools available from various vendors and resource sites. [FontForge a free font editor available from fontforge.sourceforge.net may have capabilities to perform the actions described below]

For simplicity we'll assume arial as a base font here, but you can choose whatever font you wish to incorporate in your app.

The comments in the .inc file are there to help you ensure that you gather together all the characters that your app needs so that you can make appropriate font provision.

Let us use the sample code in **Error! Reference source not found.** to check character availability.

We can see that character codepoints appearing in the comments of the snippet are:-

U+D1, U+4E2D, U+6587

So... we need to check if these are present in our Arial font. If they aren't then we may make the decision to add them, creating a custom font, based on Arial, for the project.

The Character Map tool in Microsoft Windows will enable us to look for characters within fonts that are on windows, but with a font editor it would be better to open the arial.ttf file that we are using (for sake of example) for the app, using the font editor to look inside it.

With can see that the character of codepoint U+D1 is the 'Ñ' character as in ESPAÑOL and that it is available in Arial font.

The character of codepoint U+4E2D is the Chinese ideogram 中. The simsun.ttf font contains a good many Eastern Asian characters/ideograms so, knowing the codepoint, we can open simsun.ttf in a font editor and look for it at codepoint U+4E2D within the font.

The character of codepoint U+6587 is the Chinese ideogram 文. The simsun.ttf font contains a good many Eastern Asian characters/ideograms so, knowing the codepoint, we can open simsun.ttf in a font editor and look for it at codepoint U+6587 within the font.

The font editor can then be used to add a new glyph to a base font for each of the glyph characters that we need and have found in simsun.ttf.

Example steps:-

- Take a copy of arial.ttf and rename it charial.ttf (or whatever you want to call your customised font).
- Open charial.ttf with the font editor. Open simsun.ttf with the font editor.
- Within the charial font, navigate to the end of the font and insert/add two new glyphs.
- Within the simsun.ttf font navigate to the glyph at codepoint U+4E2D, select it and copy it (e.g. Ctrl-C)
- Within the charial font, navigate to the first of the empty glyphs that you added, select it, and Paste (e.g, Ctrl-V) the glyph copied from simsun into it.
- Repeat this for the glyph at U+6587 codepoint.
- You may quite have to manually set the codepoint mapping of these new glyphs within the charial font as the editor may not have tranfered the mapping property of the glyphs with the glyphs themselves.
- Save the charial font as charial.ttf from the editor.

You'd now have a custom font file charial.ttf which has all the original characters of arial.ttf and also has the two Chinese ideograms that you added from the simsun font.

With charial selected as current font in your app, you would be able to display 中文 by out-text ing the cucSystemWords_00001_003 string, or more conveniently,
cucSystemWords[wLANG][SelectedLanguage]

Hardware Specification

Item	Description
CPU	Freescale i.MX 286, running at 454 MHz
NAND FLASH	128MB
SDRAM	128MB
DISPLAY	QVGA TFT (320 x 240) with Backlight
USB	Dual Mode Host/Device USB
RS232	1 port
RS485	None
CAN2.0B	2 ports (one isolated)
Buzzer	1
Keypad	5 keys with Backlight
Power	10-36VDC with protection

Connector Pinout/Cable Harness

Cable 1

PIN	Wire Colour	Signal Name
1	BLACK	GND
2	RED	PWR
3	BROWN	RLA1
4	VIOLET	RLA2
5	GN/YL	CAN2_GND
6	PINK	CAN2_PWR
7	GREY	CAN2_H
8	BLUE	CAN2_L
9	YELLOW	RLA3
10	ORANGE	RLA4
11	BLUE	CAN1_L
12	WHITE	CAN1_H

Cable 2

PIN	Wire Colour	Signal Name
1	BLACK	AN1
2	BROWN	AN2
3	RED	AN3
4	ORANGE	AN4
5	YELLOW	AN5
6	GREEN	AN6
7	BLUE	AN7
8	PINK	DIG_IN1
9	PK/BK	DIG_IN2
10	VIOLET	TACH
11	GN/YL	RS232RX
12	YL/RD	RS232TX